PARALLEL PROGRAM CORRECTNESS THROUGH REFINEMENT*

Thomas W. Doeppner, Jr.
Program in Computer Science
Box F
Brown University
Providence, R.I. 02912

ABSTRACT: We develop a theory for the correctness of asynchronous parallel programs. A program is considered correct if its behavior is in some sense similar to that of an abstract version of the program. We discuss various criteria for this similarity. We then concentrate on one of them and develop a technique for showing that a parallel program is correct with respect to this criterion.

1. INTRODUCTION

The benefits of a top-down approach to the design of programs are well known (Dijkstra [2]). The use of such techniques for the correctness of sequential programs makes correctness proofs simpler (Dijkstra [2], Gries [5], Infante and Montanari [6], for example). We will develop some techniques for using this approach for proving the correctness of asynchronous parallel programs.

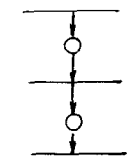
The correctness of parallel programs has been studied in Keller [7], Lamport [9], and Owicki and Gries [12]. Our approach bears some resemblance to that of Rosen [13]. His use of the Church-Rosser property and equivalent states is similar to the notions of consistency that we develop, but he does not explicitly take advantage of program "structure". Part of approach is also related to that of Lipton [11], although his emphasis is "bottom-up" while ours is "top-down".

As the first step in the development of a parallel program, we envision an "idealized" version in which "complex" operations are assumed to be performed in an indivisible manner. For example, see Figure 1a. Here we have a program, called EQ_0, for appending item i to the end of a queue. Several processes may attempt to perform this operation simultaneously, so we present an idealized version in which the operation is assumed to be performed instantaneously. As far as correctness is concerned, it is easy to verify that the predicate "next(end)=null" is invariant, i.e. true for all reachable states.

As the next step in the development of a parallel program, we break up sections of the program that were previously assumed to be indivisible into "more reasonably sized pieces". That is, we use operations that are closer to those that some processor might actually perform "instantaneously".

*This work was supported by the National Science Foundation through grant GJ-42627 and by Bell Laboratories, Murray Hill, N.J. while the author was at Princeton University.

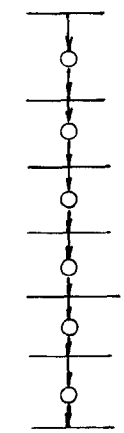
initially end is the address of a cell such that next(end)=null
private i



next(end) ← i
next(i) ← null
end ← i

Here i represents the address of some data cell, and next is a field within the cell which takes on values that refer to other cells.

Figure 1a.



initially S=1

P: when S 0
do S ← S-1

next(end) ← i

next(i) ← null

end ← i

V: S ← S+1

Figure 1b.

Consider the program of Figure 1b, which we call EQ_1. Here we have broken up the operation of appending item i onto the queue into three operations, and have used Dijkstra's P and V operations on the semaphore S to ensure that only one process can execute these operations during any one period of time. The predicate "next(end)=null", that was invariant in EQ_0, is no longer invariant, although intuitively the two parallel programs are, at the least, very similar.

There are two approaches to the correctness of EQ_1. The first is to modify the original predicate so that it is invariant in EQ_1. The other is to weaken our criteria for correctness, so that we consider a program to be correct even if the desired predicate is not invariant.

The first approach, although plausible, has a few difficulties associated with it. Modifying the predicate so that it becomes invariant in the

"expanded" parallel program may be fairly difficult to do. Even if an invariant predicate can be found, its relation to the original predicate might not be clear. Since we are using a top-down approach to the design of a program, the relationship between the programs at different levels of abstraction should be well-defined.

Let us consider the second approach to the correctness of EQ_1. It is clear that the predicate "next(end)=null" is true for all reachable states in which no processes are "between" the P and V operations. Hence EQ_1 is actually simulating EQ_0 in that the composite effect of transitions t_1 through t_5 of EQ_1 is exactly that of transition t of EQ_0. If we can prove that, in some sense, the execution of EQ_1 is "guided" by the execution of EQ_0, then the predicate that is invariant in EQ_0 can be used as a "weaker" indication of the correctness of EQ_1. What we need then is a formal method of propagating the correctness of a program at one level to the programs at lower levels as we descend through the various levels of abstraction.

2. MODEL AND NOTATION

Our model of a parallel program is similar to that presented in Keller [7]. We present the model in a slightly different way, as we want to put more emphasis on the individual processes.

A parallel program (or system) is a 4-tuple $\langle Q,V,\Pi,\Sigma\rangle$. Q is a possibly infinite set of states, V is a finite set of variable names, Π is a possibly infinite set of processes, and Σ is a finite set of transition names. Each state of Q is a vector, the components of which are named, not necessarily uniquely, by elements of V. Each process of Π consists of a name π, and a partial function pf_π mapping Q into Q.

Certain components of the state vector have unique names from V, and are referred to as shared variables. The remainder of the components do not have unique names, and are referred to as private variables. Each private variable can only be modified by a unique pf, and hence is associated with some particular process.

We qualify references to private variables by appending the process name in parentheses after the name of the variable. For example, if c is the name of a private variable, then the particular variable named c that is "accessible" by process π is called $c(\pi)$.

For any shared variable b or private variable $c(\pi)$, the value of the variable in state q is referred to as q.b or $q.c(\pi)$, respectively.

Associated with each process π is a private variable $i(\pi)$ called the instruction pointer. This variable can only take on values from a finite set of places.

There is a natural way in which we can name the ordered pairs of states that are related by the various pf's. We define a transition to be a set of ordered pairs of states, each related by some pf, such that the values of instruction pointers are equal in all states which are first in an ordered pair, and are also equal in all states which are second in an ordered pair. Each such transition is given a name from Σ. We assume a one-one correspondence between all transitions and Σ, and hence consider Σ to represent the set of all transitions.

Each transition defines a relation on the states. If t is a transition, and q_1 and q_2 are two states related by t, we then write this as

$$q_1 \xrightarrow{t} q_2$$

Only one of the instruction pointers can differ in value between the two states, as the transition relation is a subset of the relation specified by some pf. If π is the process name associated with the particular instruction pointer involved, then we say that π executed the transition t. If need be, we modify our notation and explicitly mention π as follows.

$$q_1 \xrightarrow{t(\pi)} q_2$$

If q_1 and q_2 are related by some unspecified transition we write $q_1 \rightarrow q_2$, i.e., $\rightarrow$ is the relation on $Q\times Q$ which is the union of all the pf's.

Letting e denote the null transition, we write $q \xrightarrow{e} q$ for all states q. If, for all states q_1, q_2, and q_3

$$q_1 \xrightarrow{x} q_2 \ \wedge \ q_2 \xrightarrow{t} q_3$$

for $x\in\Sigma^*$ and $t\in\Sigma$, we then write

$$q_1 \xrightarrow{xt} q_3$$

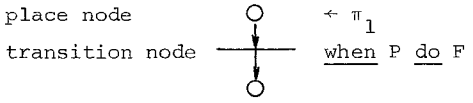
If there exists an $x\in\Sigma^*$ such that $q_1 \xrightarrow{x} q_2$, we write $q_1 \xrightarrow{*} q_2$. If we need to specify to which parallel program we are referring, say S_j, we write, for example,

$$q_1 \xrightarrow[S_j]{t} q_2 \quad \text{or} \quad q_1 \xrightarrow[j]{t} q_2.$$

We will always specify some state, written as q_0, to be the start state. If there exist states q_1 and q_2 such that $q_1 \xrightarrow{x} q_2$, then we say that x is a valid transition sequence. If q_1 is the start state, then we say that x is a valid initial transition sequence.

If there exists a valid transition sequence x such that $q_1 \xrightarrow{x} q_2$, then we say that q_2 is reachable from q_1, or just reachable if q_1 is the start state q_0.

Following Keller [7], we present a graphical interpretation of our model. We use a bipartite directed graph, with one class of nodes, written as horizontal lines, representing transitions, and the other class of nodes, written as circles, representing places, i.e. the values of instruction pointers. See Figure 2.



Process π_1's instruction pointer has the value of the indicated place.

Figure 2.

There is an arc from a place a to transition t and from transition t to place b iff there exist two states q_1 and q_2 such that $q_1 \xrightarrow{t} q_2$, and in q_1 some instruction pointer has the value a, while in q_2 it has the value b. If there is an arc from one node to another, we say that the former is an input node of the latter, and the latter is an output node of the former.

In any particular state of a parallel program, the individual processes are thought of as dwelling at the place nodes representing the values of their

respective instruction pointers. We often attach to transition nodes information specifying the relation that the transition represents. For example, for a predicate P and a function F we write "When P do F" to mean that if there is a process π dwelling at the input place of the transition and P is true (when this is true, we say that π is enabled for the transition), then π may "move" so as to dwell at the output place of the transition, and at the same instant the state is further modified as specified by F. P is called the enabling predicate of the transition, and F is called the action function of the transition. See Figure 2.

If the disjunction of the enabling predicates of the output transitions of a place is identically true, then the place is called a non-synchronizing place, and the output transitions are called non-synchronizing transitions. Otherwise, the place is called a synchronizing place, and the output transitions are called synchronizing transitions.

We often include in our graphical model transition nodes with no input places, representing "creators of processes", called entrance transitions, and include transitions with no output places, representing "annihilators of processes", called exit transitions. See Figure 3. An entrance transition is always enabled, and hence allows the introduction of an unbounded number of processes into the system. Exit transitions, when executed, effectively destroy a process.

Figure 3.

These special transitions are not intended to increase the "power" of the model. They are just a convenient shorthand. The creation of processes could just as well be represented by infinite "pools" of processes dwelling at certain places, just biding their time until they are allowed to proceed, and destruction of processes could be handled in a similar fashion.

A typical way of showing a parallel program to be "correct" is to prove that all reachable states satisfy some predicate. Following Keller [7], we give the following definition.

DEFINITION: Let $\langle Q,V,\Pi,\Sigma\rangle$ be a parallel program. A unary predicate J on Q is said to be q-invariant if

$$(\forall q' \in Q)\ q \overset{*}{\rightarrow} q' \Rightarrow J(q').$$

If q is omitted, then we mean q_0-invariant.

Proving a predicate to be invariant can be very difficult, if done directly. It is most often easier, if one wants to prove J invariant, to prove a stronger version of J to be inductive, as defined below (also following Keller [7]).

DEFINITION: Let $\langle Q,V,\Pi,\Sigma\rangle$ be a parallel program. A unary predicate K on Q is said to be q-inductive if

$$K(q) \wedge (\forall q_1,q_2 \in Q)\,(K(q_1) \wedge q_1 \rightarrow q_2) \Rightarrow K(q_2)$$

If q is omitted, then we mean q_0-inductive.

An important property of a parallel program is that of deadlock-freedom. Following Doeppner and Keller [4], we formalize this property.

DEFINITION: We say that a process π is dead in state q if π has not executed an exit transition and there exists no state q' such that $q \overset{*}{\rightarrow} q'$ and π is enabled in q'. We write this condition as $dead_\pi(q)$.

DEFINITION: A parallel program is said to be deadlock-free if for all processes π, $\neg dead_\pi$ is q_0-invariant.

3. EXPANSION

When developing a parallel program, it is convenient at first to treat certain complex operations as indivisible. When more detail is required, the complex operations can be split up into less complex, component operations, with the execution of these components interleaved with the execution of operations of other processes. We call this expansion.

For example, see Figure 4. In Figure 4a, we have two processes, with one process performing two additions in one step. In Figure 4b, we have expanded this "complex" operation, and introduced new variables to control the execution of the expanded operation. Correctness of such expansions will be considered in the next section. In this section we formally explore the notion of expansion.

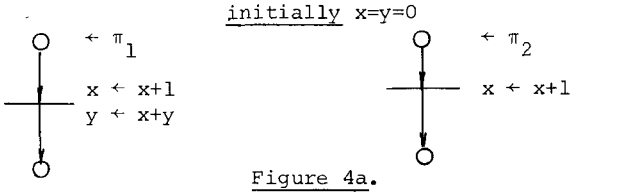


Figure 4a.

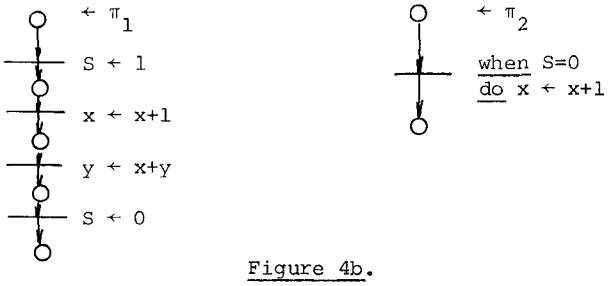


Figure 4b.

Let us consider a parallel program S_i. We wish to refine transition t of S_i, resulting in a new parallel program S_{i+1}. We write this act of expansion as

$$S_i \xrightarrow{t} S_{i+1}.$$

If t is understood or not important, we omit the t. We call transition t the prototype of the expansion, and we call the process which contains t the expansion process.

Let p_1 and p_2 be the input and output places of t, respectively. In the graphical interpretation of S_i, there is a path from p_1 to p_2 containing only one node, t. By expanding S_i, we replace this single path with a subgraph, called the expansion subgraph of S_{i+1}, containing new place nodes called expansion places, and new transition nodes called expansion transitions. Σ_{i+1} is formed from Σ_i by replacing t with the expansion transitions. p_1 is called the input place of the expansion subgraph, and p_2 is called the output place of the expansion subgraph.

Referring to Figure 4, process 1 is the expansion process. Transition t of Figure 4a is the prototype of the expansion. It is replaced by the subgraph that is shown in Figure 4b.

We will find it necessary to relate the transition names of the expanded system to those of the unexpanded system. We define a homomorphism from the transition names of the expanded system to those of the unexpanded system. For non-expansion transitions, the mapping is just the identity mapping. For expansion transitions, we will map all but one into the empty string, and the remaining transition will map into the prototype of the expansion. The choice of which one to map into the prototype is somewhat arbitrary. We choose the last transition of an expansion sequence, mainly because in this way only complete expansion sequences will map into the prototype.

More formally, we define the homomorphism μ_{i+1} mapping Σ_{i+1} onto Σ_i, where

$\mu_{i+1}(s) = s$ if s is not an expansion transition.

$\mu_{i+1}(t_n) = t$ if t_n is an expansion transition, the output place of which is the output place of the expansion subgraph, and t is the prototype of the expansion.

$\mu_{i+1}(t_i) = e$ for all other expansion transitions t_i, where e is the empty string.

This mapping is illustrated in Figure 5.

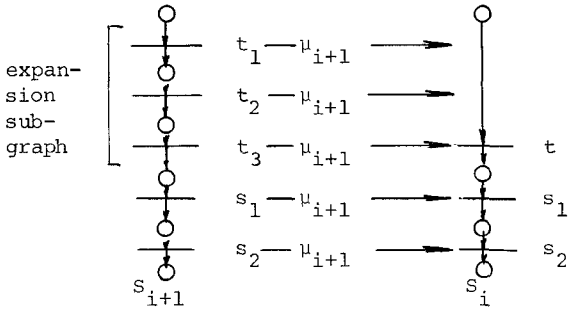


Illustration of the mapping μ_{i+1}.

Figure 5.

We augment the state set of S_i by introducing new (non-instruction pointer) variables in the state vectors, where these variables take on some set of values. Let Q_i denote the state set of S_i and Q_{i+1} denote the state set of S_{i+1}. We define a partial function η_{i+1} mapping Q_{i+1} onto Q_i, where $\eta_{i+1}(q)$ is the state q' in Q_i such that the values of each of the variables in q' are the same as those in q. η_{i+1} is not defined when the expansion process's instruction pointer has the value of an expansion place. We attach to the transition nodes replacing the expansion transition information specifying state relations, i.e., enabling predicates and action functions.

We always use q_0 to denote the start state of a system. Although this is somewhat ambiguous, the system of which q_0 is the start state should be clear from context.

Other processes are also possibly modified by the addition of new variables, and the corresponding pf's are extended over the larger domain. We restrict the modification of other processes so that if π' is not the expansion process, then

$$q_1 \xrightarrow[i+1]{s(\pi')} q_2 \Rightarrow \eta_{i+1}(q_1) \xrightarrow[i]{s(\pi')} \eta_{i+1}(q_2)$$

We call this restriction faithfulness.

The expansion subgraph defines a set of paths from p_1 to p_2, which we wish to treat collectively.

DEFINITION: The set of expansion sequences T of S_{i+1} is the set of all transition sequences from paths from the input place to the output place of the expansion subgraph of S_{i+1} that can appear as a subsequence of a valid transition sequence.

We want to ensure that in some sense expansion sequences "mimic" the operation performed by the prototype transition. We make certain that this is the case when non-expansion transitions are not interleaved with expansion transitions.

DEFINITION: A set of expansion sequences T is said to be accurate if for all sequences $t_1,\ldots,t_n$ from T and all states q_1 and q_2 from Q_{i+1}

$$q_1 \xrightarrow[i+1]{t_1 \ldots t_n} q_2 \Rightarrow \eta_{i+1}(q_1) \overset{t}{\underset{i}{\rightarrow}} \eta_{i+1}(q_2)$$

where t is the prototype transition (note that this definition only concerns the composite behavior of expansion transitions; there is no mention of the effects of interleaved execution with transitions of other processes).

We should contrast faithfulness with accurate expansions. Faithfulness is a restriction on non-expansion transitions, while accuracy is a similarly motivated restriction on expansion transitions. As accuracy seems to be the more difficult to verify, we stress this difficulty in the remainder of our work by explicitly requiring an expansion to be accurate, while we implicitly assume it to be faithful.

What we have described concerns only the refinement of a single transition, which we shall refer to as a single expansion. We are usually concerned with a sequence of expansions, i.e., $S_0 \Longrightarrow S_1$, $S_1 \Longrightarrow S_2, \ldots, S_{i-1} \Longrightarrow S_i$. We abbreviate our notation as $S_0 \Longrightarrow S_1 \Longrightarrow S_2 \Longrightarrow \ldots \Longrightarrow S_i$, and call such a sequence of expansions a multi-expansion, or simply expansion. When referring to a multi-expansion, our use of the terms expansion process, prototype transition, etc. should be interpreted as referring to the "last" expansion, i.e. $S_{i-1} \Longrightarrow S_i$ of the aforementioned multi-expansion.

We have now developed our idea of an expansion and presented a notation. The remainder of the paper mainly concerns what happens when the execution of non-expansion transitions is interleaved with that of expansion transitions.

4. CONSISTENCY

Suppose that a system S_0 is "correct", and system S_i is the result of expanding S_0. We want to develop criteria on the basis of which we can decide whether the correctness of S_i can be inferred from the correctness of S_0. The criteria which we discuss are based on the notion of S_i being able to "simulate" S_0.

There are two parameters involved in this simulation. Very roughly, the first parameter concerns what we want to be "preserved" by the expansion. We can use the strict requirement that state

reachability be preserved, i.e. that what is true in S_0 always be true in S_i. We can also use the weaker requirement (assuming that S_0 is judged correct because a predicate W is inductive) that only W need be preserved.

The second parameter concerns how closely S_i should follow S_0. Again we have two choices. The first is the strict requirement that S_i can only "deviate" from S_0 when S_i is executing expansion transitions. The second is the weaker requirement that S_i be allowed to "stray" from the path of S_0, as long as it can always "get back" onto the path of S_0.

We first mention three intuitive ideas which we often desire to be incorporated as constraints in our correctness criteria. The first and third constraints we always require. The second, because it is not essential to the proofs of later theorems, we present as an "optional" requirement. These ideas are similar in spirit to those of Lipton [10] in his definition of "simulate".

Our first constraint is that we want both the unexpanded and expanded systems to be deadlock-free. If it is desired to have a process "terminate", this can be accomplished by the use of an exit transition.

The second constraint, which we often place on a "correct" expansion, is what we call our boundedness constraints. We want to bound the amount of time a process spends executing the expansion transitions, as opposed to just constraining a process to spend a finite amount of time executing these transitions. The essence of the situation that we wish to avoid is expressed in Figure 6. If

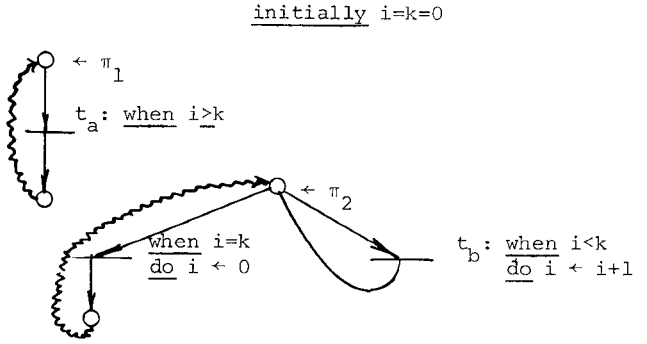


Figure 6.

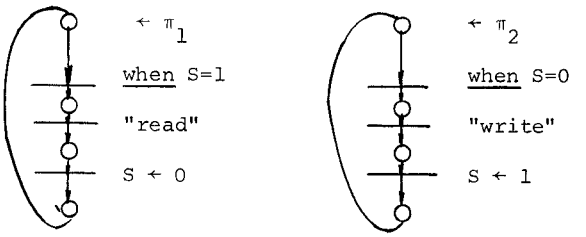
process 1 is the expansion process, and t_a is an expansion transition, then t_a will always be able to be executed after a finite period of waiting, but this waiting period will increase without bound after each successive execution of t_a. In a "real" system, such a situation would probably result in a continuing degradation of performance. Hence we usually want to ensure a bounded period of waiting.

Similarly, if process 2 is regarded as the expansion process and t_b an expansion transition, then after each successive execution of an expansion sequence containing t_b, the expansion sequence will "grow" longer, without bound. Hence we usually want to bound the length of expansion sequences.

The third constraint is the requirement that, in some sense, the expanded system can do everything that the unexpanded system can do. For example, consider the well-known reader-writer

idealized reader-writer

Figure 7a.



"bad" solution to reader-writer problem

Figure 7b.

problem (Courtois et al. [1]). In Figure 7a, we have an "idealized" solution to it. In Figure 7b, we have expanded the solution, but the expansion requires that the readers and writers alternate. This we do not wish to consider correct, as the expanded system should have "at least as many degrees of freedom" as the unexpanded system.

The following definitions are useful in relating the states of the various systems.

DEFINITION: We let $\eta_k^*(q)$ represent $\eta_1(\eta_2(\ldots\eta_k(q))\ldots))$ if each of the η_i is defined.

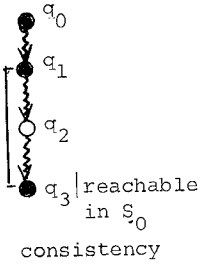
DEFINITION: A state q of S_i is said to be conceivable in S_j for i>j, if $\eta_{j+1}(\ldots\eta_{i-1}(\eta_i(q))\ldots)$ is defined.

Conceivability is an important concept and should be emphasized. A state of S_i is conceivable in S_j if the instruction pointers of all processes take on the values of places that exist in S_j. In such states we can make direct comparisons of the two systems. Our correctness criteria will be based on reaching these conceivable states, and showing that certain properties of the states of S_j also hold for the states of S_i which are conceivable in S_j.

Our first correctness criterion is fairly strict, and useful mainly in systems with trivial interaction among processes. Referring back to the "parameters of simulation", we are using their strict values. That is, if $S_0 \Longrightarrow S_1 \Longrightarrow \ldots \Longrightarrow S_i$, then S_i should always be capable of reaching a state conceivable in S_0, and that all reachable states of S_i that are conceivable in S_0 should be such that their images under η_i^* are reachable in S_0.

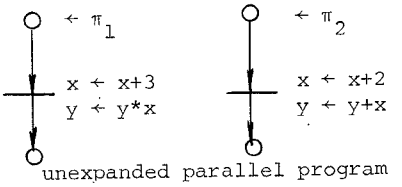
This correctness criterion is diagrammed in Figure 8. Circles represent states of S_i, and solid circles represent states that are conceivable in S_0. The wavy lines represent transition sequences, and the bracket means that between states q_1 and q_3, the system enters no state that is conceivable in S_0. The interpretation of the diagram is then that if S_i reaches a state (q_2) that is not conceivable in S_0, then the next state reached that is conceivable in S_0 will be "reachable" in S_0.

An example of an expansion that is correct under this criterion is given in Figure 9. The expanded system is effectively behaving exactly as the unexpanded system.



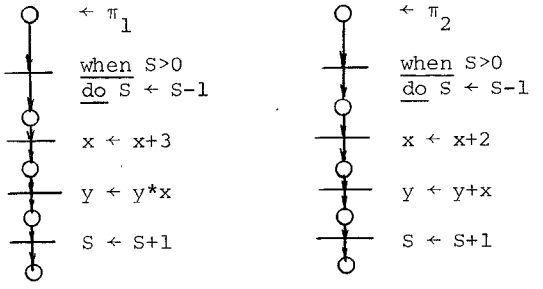
consistency

Figure 8.



unexpanded parallel program

Figure 9a.



a consistent expansion

Figure 9b.

In the following, Σ_0 and Σ_i denote the respective sets of transition names of S_0 (the unexpanded system) and S_i (the expanded system). n is understood to be a natural number. ? denotes "undefined". $R_j(q_0)$, the "reachable set" is defined to be

$$\{q \mid q_0 \overset{*}{\underset{j}{\rightarrow}} q\}$$

DEFINITION: An expansion $S_0 \longrightarrow\!\!\!\!\rightarrow S_1 \longrightarrow\!\!\!\!\rightarrow \ldots \longrightarrow\!\!\!\!\rightarrow S_i$ is boundedly consistent if $S_0, S_1, \ldots, S_i$ are deadlock-free and

$$(\exists n)(\forall q \in R_i(q_0))$$
$$\eta_j^*(q) \neq ? \Rightarrow \eta_i^*(q) \in R_0(q_0)$$
$$\eta_i^*(q) = ? \Rightarrow ((\exists q' \in Q_i)(\exists x \in \Sigma_i^{*n})(q \overset{x}{\underset{i}{\rightarrow}} q' \wedge \eta_i^*(q) \neq ?))$$

If we replace "$(\exists x \in \Sigma_i^{*n})$" with "$(\exists x \in \Sigma_i^*)$", then an expansion satisfying the criterion is said to be consistent.

Ordinarily, in an expansion we are not concerned so much with preservation of reachability as with the preservation of some weaker predicate W. So we weaken the first "parameter of simulation". That is, we modify the consistency definition so that it only need be that W is true of reachable states that are conceivable in S_0.

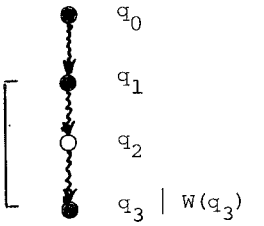
DEFINITION: An expansion $S_0 \longrightarrow\!\!\!\!\rightarrow S_1 \longrightarrow\!\!\!\!\rightarrow \ldots \longrightarrow\!\!\!\!\rightarrow S_i$ is boundedly W-consistent if $S_0, S_1, \ldots, S_i$ are deadlock-free and

$$(\exists n)(\forall q \in R_i(q_0))$$
$$\eta_i^*(q) \neq ? \Rightarrow W(q)$$
$$\eta_i^*(q) = ? \Rightarrow ((\exists q' \in Q_i)(\exists x \in \Sigma_i^{*n})(q \overset{x}{\underset{i}{\rightarrow}} q' \wedge \eta_i^*(q) \neq ?))$$

If we replace "$(\exists x \in \Sigma_i^{*n})$" with "$(\exists x \in \Sigma_i^*)$", then an expansion satisfying the criterion is said to be W-consistent.

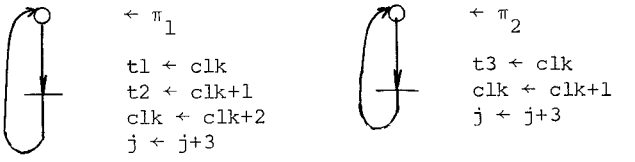
W-consistency is diagrammed in Figure 10. Here, from any reachable state of S_i that is not conceivable in S_0, the next state reached that is conceivable in S_0 will be such that W is true.

An example of an expansion that is W-consistent



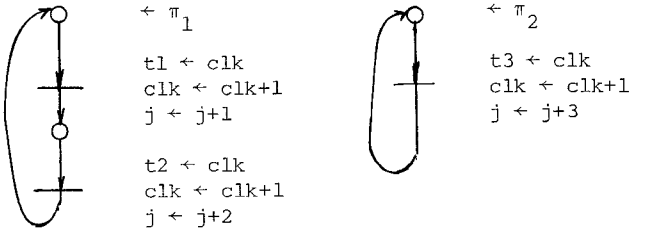
W-consistency

Figure 10.



unexpanded parallel program

Figure 11a.



a "j is a multiple of 3"-consistent expansion

Figure 11b.

but not consistent is shown in Figure 11. Here W is the predicate "j is a multiple of 3". The expansion is clearly W-consistent, but it is not consistent since in the unexpanded system t1 and t2 will always have consecutive values, but in the expanded system this is not necessarily so.

Consistency and W-consistency both require a certain "structuredness" of an expansion. That is, the expanded system is only allowed to behave significantly differently than the unexpanded system when it is "executing" an expansion sequence. This structure makes proofs of consistency and W-consistency relatively easy, as we shall see in the next section. However, we may still want to consider some expansions to be correct even if they do not display this structure.

So we weaken the second parameter and strengthen the first parameter of simulation. That is, we weaken our first correctness criterion in another way so that we do not require all reachable states of S_i that are conceivable in S_0 to be reachable in S_0. What we require is that there always be reachable a state from which direct simulation of S_0 is possible.

DEFINITION: An expansion $S_0 \longrightarrow\!\!\!\!\rightarrow S_1 \longrightarrow\!\!\!\!\rightarrow \ldots \longrightarrow\!\!\!\!\rightarrow S_i$ is boundedly semi-consistent if $S_0, S_1, \ldots, S_i$ are deadlock-free, and there exists an integer n such that for all reachable states q of S_i, there is a valid transition sequence $x \in \Sigma_i^{*n}$ and a state q' such that

1) $q \overset{x}{\underset{i}{\rightarrow}} q'$

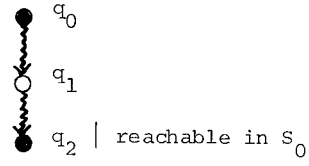
2) $\eta_i^*(q')$ is reachable in S_0

3) if $\eta_i^*(q') \overset{*}{\underset{0}{\rightarrow}} q_a$, then there is a state q_b in S_i such that $\eta_i^*(q_b) = q_a$, and there is a transition sequence $y \varepsilon \Sigma_i^{*n}$ and a state q' such that $q' \overset{y}{\underset{i}{\rightarrow}} q_b$.

If we replace "Σ_i^{*n}" with "Σ_i^*", then an expansion satisfying the criterion is said to be semi-consistent.

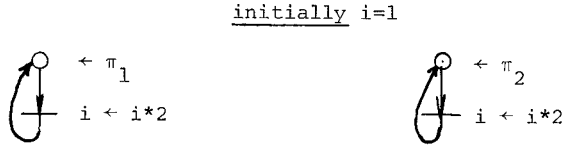
Semi-consistency is diagrammed in Figure 12. Here, from any state of S_i, there is reachable a state that is conceivable in S_0 and "reachable" in S_0.

In Figure 13, we have an example of a system that is semi-consistent, but neither W-consistent nor consistent. In all states reachable in the unexpanded system, i is a power of two. In the expanded system, there certainly are reachable states in which i is not a power of two. However, from any such state there is reachable a state in which i is a power of two. From this state, direct simulation of the unexpanded system is possible.



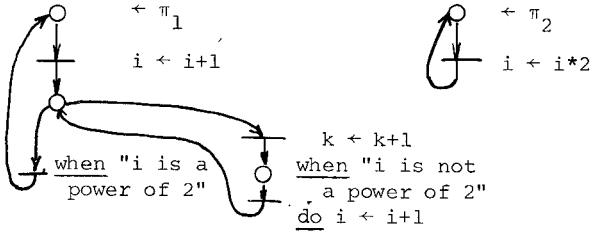
semi-consistency

Figure 12.



unexpanded parallel program

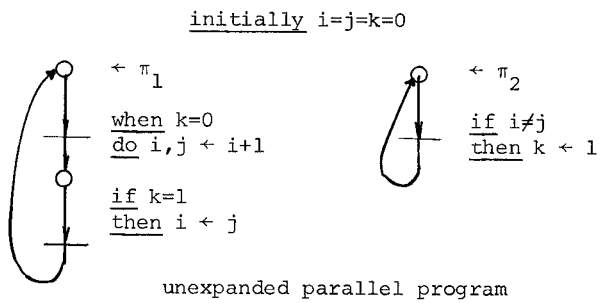
Figure 13a.



a semi-consistent expansion

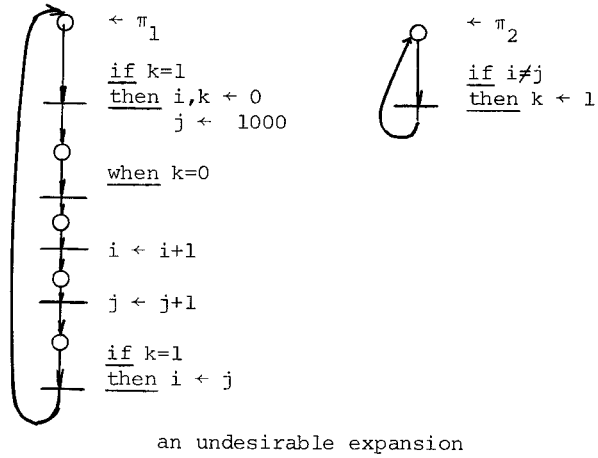
Figure 13b.

In certain cases, when W is inductive in S_0, we can combine the previous two correctness criteria, i.e., we weaken both parameters of simulation. The combined criterion is that from any reachable state of S_i, there is reachable a state which is conceivable in S_0, and in which the predicate W is true. Presumably, this would mean then that from this state, direct simulation of S_0 is possible, and hence every time S_i enters a state conceivable in S_0, W would be true. Unfortunately, this is not always the case. Although S_0 and S_i may both be deadlock-free, it is possible that a state q of S_i that is conceivable in S_0 may be such that $dead_\pi(\eta_i^*(q))$.



unexpanded parallel program

Figure 14a.



an undesirable expansion

Figure 14b.

For example, see Figure 14. In the unexpanded system, the predicate "i=j" is clearly invariant. In the expanded system, it is possible for i and j to become unequal. Once this occurs, they will only again be equal when an instruction pointer takes on the value of p_1. But in such a state, k will be 1. This is clearly a deadlock situation in the unexpanded system. For the process to proceed in the expanded system, it must set i and j to unequal values. Such an expansion one surely does not want to consider correct.

Because of this problem, we require the reachability of a state in S_i which is such that it is conceivable in S_0, W is true, and in its image in S_0 no process is dead.

DEFINITION: An expansion $S_0 \Longrightarrow S_1 \Longrightarrow \ldots \Longrightarrow S_i$ is boundedly semi-W-consistent if S_i is deadlock-free, W is inductive in S_0, and there exists an integer n such that for all reachable states q of S_i, there is a valid transition sequence $x \varepsilon \Sigma_i^{*n}$ and a state q' such that

1) $q \overset{x}{\underset{i}{\rightarrow}} q'$

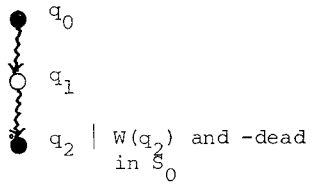
2) q' is conceivable in S_0

3) -dead_π is $\eta_i^*(q')$-invariant in S_0 for all processes π

4) if $\eta_i^*(q') \overset{*}{\underset{0}{\rightarrow}} q_a$, then there is a state q_b in S_i such that $\eta_i^*(q_b) = q_a$ and there is a transition sequence $y \varepsilon \Sigma_i^{*n}$ such that $q' \overset{y}{\underset{i}{\rightarrow}} q_b$.

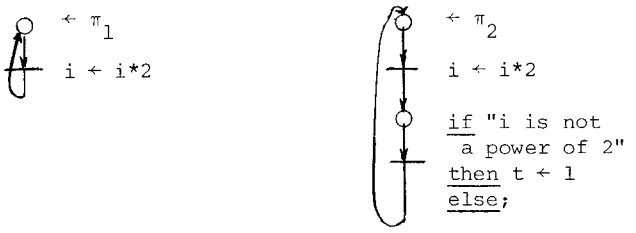
If we replace "Σ_i^{*n}" with "Σ_i^*", then an expansion satisfying the criterion is said to be semi-W-consistent.

Semi-W-consistency is diagrammed in Figure 15. This is interpreted as, from any state that is reachable in S_i, there is reachable a state that is



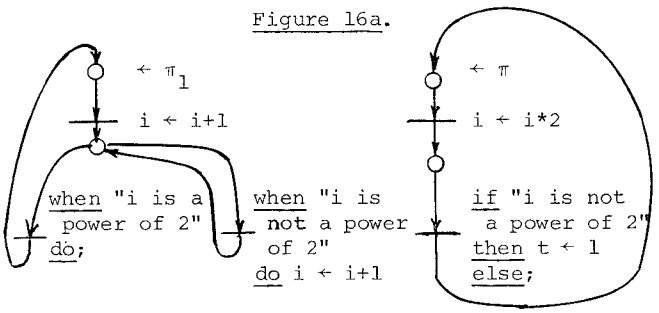
semi-W-consistency

Figure 15.



unexpanded parallel program

Figure 16a.



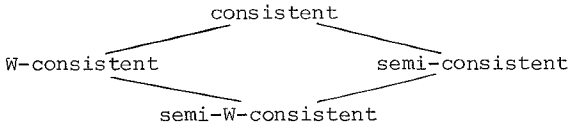
a semi-"i is a power of two"-consistent expansion

Figure 16b.

conceivable in S_0, "not dead in S_0", and such that W is true for the state.

An example of an expansion which is semi-W-consistent but neither semi-consistent, W-consistent, nor consistent is given in Figure 16. This is basically the same as the example of a semi-consistent expansion, but the setting of the variable t whenever i is not a power of two prohibits the possibility of direct simulation.

We have presented formalizations of our intuitive ideas of how a correct expansion should behave. We summarize our criteria in Figure 17, where an expansion satisfying an "upper criterion" implies that it satisfies the "lower criteria".



summary of criteria

Figure 17.

5. PROVING CONSISTENCY

In his paper on reduction [11], Lipton discusses a technique which, in the terminology and context of our present work, is a method for proving an expansion to be consistent. We develop techniques of a similar motivation for proving an expansion to be W-consistent. However, we first concentrate on consistency, developing a method similar to that of Lipton.

It is often convenient to view the component transitions of an expansion as one transition performing some desired action, and a set of other transitions performing actions that are more of a bookkeeping nature, not really affecting the progress of other processes. The former transition we call the representative of the expansion, in that with respe-t to other processes, it embodies all that is important of the expansion transitions.

The usefulness of this idea is that it provides a simple method for proving an expansion to be consistent. In developing this idea, we need the following definition.

DEFINITION: Let s be a transition of a process different from the expansion process and t_r an expansion transition. s is said to commute around t_r (in the expanded system) iff for each sequence $\langle t_1,\ldots,t_n\rangle$ from the set of expansion sequences T, all states q_1 and q_2, and for each $t_i \in \langle t_1,\ldots,t_n\rangle$, if t_i precedes t_r, then

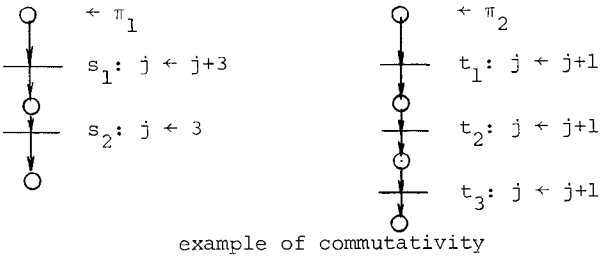
$$q_0 \xrightarrow{*} q_1 \xrightarrow{t_i s} q_2 \Rightarrow q_1 \xrightarrow{st_i} q_2$$

and if t_r precedes t_i, then

$$q_0 \xrightarrow{*} q_1 \xrightarrow{st_i} q_2 \Rightarrow q_1 \xrightarrow{t_i s} q_2$$

Transition t_r is said to be a representative of the expansion.

In Figure 18, transition s_1 commutes around t_2, but s_2 does not. Here transitions t_1, t_2, and t_3 are expansion transitions.



example of commutativity

Figure 18.

Intuititively, transition s commutes around t_r if the execution of s is in no way affected by the execution of any of the transitions in the expansion except t_r.

We need to make certain that a process cannot get "stuck" while executing expansion transitions, i.e., that the system can always reach a state which is conceivable in the unexpanded system. We formalize this constraint below.

DEFINITION: An expansion $S_0 \quad S_1$ is said to be extendable if for each transition sequence x such that $q_0 \overset{x}{\rightarrow} q_2$, there exists a y from Σ_1^* such that $q_1 \overset{y}{\rightarrow} q_2$ and q_2 is conceivable in S_0.

We now "combine" our definitions into one, resulting in a sufficient condition for consistency.

DEFINITION: If all transitions of all processes except the expansion process commute around t_r, for some expansion transition t_r, and the expansion is accurate and extendable, then the expansion is said to be interleavable.

THEOREM 1: If $S_0 \xrightarrow{t} \gg S_1$ is an interleavable expansion, then the expansion is consistent.

PROOF: We want to show two things, which together imply that the expansion is consistent:

1) if S_1 is a reachable state that is not conceivable in S_0, then after some finite number of transitions it will be in a state that is conceivable in S_0.

2) If S_1 is in a reachable state that is conceivable in S_0, then the image of this state under η_1 is reachable in S_0.

Let x be any transition sequence such that $q_0 \overset{x}{\rightarrow} q_1$. Since the expansion is interleavable, the transitions of x can be permuted into a sequence x' so that $q_0 \overset{x'}{\underset{1}{\rightarrow}} q_1$, and all transitions of each expansion sequence in x are contiguous. Suppose that q_1 is not conceivable in S_0. This means that the rightmost transitions of x' are a proper prefix of an expansion sequence. Since the expansion is extendable, x' can be extended forming the transition sequence z = s'y. It is now true that

$$q_0 \overset{x'}{\underset{1}{\rightarrow}} q_1 \overset{y}{\underset{1}{\rightarrow}} q_2$$

where q_2 is conceivable in S_0. Since the expansion is accurate, it follows that

$$\eta_1(q_0) \xrightarrow[0]{\mu_1(z)} \eta_1(q_2)$$

which proves the theorem.

There are examples of the use of the preceding idea in Lipton [11]. In particular, he shows (in the terminology and context of our present work) that Dijkstra's P and V primitives, as used to implement mutual exclusion, form consistent expansions.

We now concentrate on proving an expansion to be W-consistent. We first consider the case of a single expansion. Later, we generalize these results, allowing multi-expansions.

Since interleavability implies consistency, we weaken the idea of interleavability into something that implies W-consistency. Suppose that when a transition sequence is permuted as in the definition of a transition commuting around t_r, the new sequence does not take the system into the same state as the previous sequence. But if the desired predicate W is still true in this new state, then we can still show the system to be W-consistent. We postulate a new set of transitions to account for the differences in states arising from permuting the transition sequences.

DEFINITION: For system S_i, let L represent the set of all partial functions mapping Q_i into Q_i such that for each $\ell \varepsilon L$ and each $q \varepsilon Q_i$, if $\ell(q)$ is defined, then $W(q) \Rightarrow W(\ell(q))$, and the value of each instruction pointer in q is the same as the corresponding instruction pointer in $\ell(q)$. We call each member of L a W-preserving residual.

NOTE: Although not transitions in the sense that they are not part of the parallel program, we will use W-preserving residuals as if they were part of the program. In order to avoid confusion, if two states of S_i are related by a transition sequence that contains W-preserving residuals, we will write

$$q_1 \overset{x}{\underset{i}{\rightarrow}} q_2$$

and say that x is a W-valid transition sequence. We extend μ_i to map W-preserving residuals into the empty string.

DEFINITION: If there exists a W-valid transition sequence x such that

$$q_0 \xrightarrow{x} q_1$$

we say that q_1 is W-reachable.

If a transition sequence of S_{i+1} is such that whenever an expansion sequence is executed, the execution of the expansion sequence is not interrupted by the execution of any other transitions, then this transition sequence is effectively "mimicking" a transition sequence of S_i, with the prototype transition replacing the expansion sequences.

DEFINITION: A W-valid transition sequence x of S_{i+1} is said to be performable in S_i if it is the case that each occurrence of an expansion sequence in x is a substring of x.

W-preserving residuals will be used for proving that W is true for states of an expanded system. Our general plan for showing an expansion to be W-consistent will be as follows. Given any valid transition sequence of an expanded system leading to a state q that is conceivable in the unexpanded system, we will show that there is another W-valid transition sequence leading to q, which is performable in the unexpanded system. Assuming the expansion to be accurate, and since W is inductive in the unexpanded system and W-preserving residuals do not falsify W, it will follow that W(q) must be true, and hence the expansion is W-consistent.

We now define a class of transition sequences which are "equivalent, modulo W-preserving residuals" to transition sequences performable in S_0.

DEFINITION: A valid transition sequence x of S_1, such that $q_1 \overset{x}{\underset{1}{\rightarrow}} q_2$ where q_1 and q_2 are conceivable in S_0, is said to be W-compressible if there exists a W-valid transition sequence y such that $q_1 \overset{y}{\underset{1}{\rightarrow}} q_2$, and y is performable in S_0.

We broaden our notion of extendable to ensure that in system S_i, states conceivable in S_0 are always reachable by a finite number of transitions.

DEFINITION: An expansion $S_0 \rightarrow\!\!\!\!\gg S_1 \rightarrow\!\!\!\!\gg \ldots \rightarrow\!\!\!\!\gg S_i$ is said to be extendable to S_0 if for each transition sequence x such that $q_0 \overset{x}{\underset{i}{\rightarrow}} q_2$, there exists a y from Σ_i^* such that $q_1 \overset{y}{\underset{i}{\rightarrow}} q_2$ and q_2 is conceivable in S_0.

If an expansion is extendable, then any valid transition sequence can be extended so that it takes the system into a state that is conceivable in the unexpanded system. We formalize this notion as follows.

DEFINITION: Let $S_0 \rightarrow\!\!\!\!\gg S_1 \rightarrow\!\!\!\!\gg \ldots \rightarrow\!\!\!\!\gg S_i$ and let x be a transition sequence such that $q_1 \overset{x}{\underset{i}{\rightarrow}} q_2$. To S_0-extend x is to append to x a transition sequence $y \varepsilon \Sigma_i^*$ so that $q_1 \overset{xy}{\underset{i}{\rightarrow}} q_3$ and q_3 is conceivable in S_0.

We are now able to weaken our notion of interleavability so that only the predicate W need be "preserved" in an expansion.

DEFINITION: An expansion $S_0 \rightarrow\!\!\!\!\gg S_1$ is said to be W-interleavable if it is accurate, and each valid transition sequence of S_1 starting from a state q such that q is conceivable in S_0 and W(q) is true can be S_0-extended so that the result is W-compressible.

Related to our previous definition of a transition sequence commuting around the prototype, we define what we call W-commute, in which we allow the insertion of W-preserving residuals to "fix things

up". The result is a sufficient, but not necessary, condition for W-interleavability.

DEFINITION: Let s be a transition of a process different from the expansion process, and let t_r be an expansion transition occurring in all expansion sequences. s is said to W-commute around t_r in S_1 if

1) W is inductive in S_0, and

2) for each sequence $\langle t_1,\ldots,t_n\rangle$ from the set of expansion sequences T, all states q_1 and q_2 from Q_1 such that $W(q_1)$ is true, and for each $t_k \varepsilon \langle t_1,\ldots,t_n\rangle$,

a) if t_k precedes t_r, then

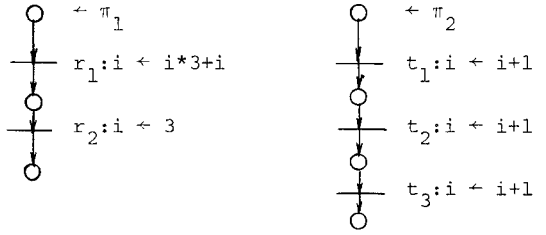
$$q_1 \xrightarrow[1]{t_1\ldots t_k s} q_2 \Rightarrow q_1 \xrightarrow[1]{\ell_1 s \ell_2 t'_1\ldots t'_j} q_2$$

where $\langle t'_1,\ldots,t'_j\rangle$ is a prefix of an expansion sequence, and ℓ_1 and ℓ_2 are W-preserving residuals,

b) if t_r precedes t_k, then

$$q_1 \xrightarrow[1]{s t_k\ldots t_n} q_2 \Rightarrow q_1 \xrightarrow[1]{t'_k\ldots t'_m \ell_1 s \ell_2} q_2$$

where $\langle t'_k,\ldots,t'_m\rangle$ is a suffix of an expansion sequence, and ℓ_1 and ℓ_2 are W-preserving residuals.



example of "i is a multiple of 3"-commutativity

Figure 19.

For an illustration of the definition of W-commutativity, see Figure 19. Here t_1, t_2, and t_3 are an expansion sequence. If we let W be the predicate "i is a multiple of 3", then r_1 W-commutes around each of the expansion transitions, but r_2 does not W-commute around any of them. Suppose that it is thought that r_2 W-commutes around t_2. Consider the transition sequence $t_1t_2r_2t_3$. If i = 0 in the initial state, this will result in a state in which i = 4. If we now consider the sequence $t_1t_2t_3r_2$, as required by the definition, this will result in a state in which i = 3. What we need is a W-preserving residual ℓ such that $t_1t_2t_3r_2\ell$ results in a state in which i = 4. But this is impossible, since ℓ would then be causing a change from a state in which W is true to a state in which W is false.

W-commutativity provides us with a sufficient condition for W-interleavability.

THEOREM 2. Let $S_0 \xrightarrow[t]{} S_1$. If all transitions of processes other than the expansion process W-commute around t_r, for some expansion transition t_r, and the expansion is accurate and extendable, then the expansion is W-interleavable.

PROOF: A straightforward induction based on the definition of W-commute.

We now proceed to show that W-interleavability provides a sufficient condition for W-consistency.

LEMMA 1: If $q_1 \xrightarrow[1]{x} q_2$, W is inductive in S_0, and x is performable in S_0, then if $W(q_1)$ then $W(q_2)$.

PROOF: x is composed of 1) transitions that exist in S_0, 2) W-preserving residuals, and 3) expansion transitions. In case 1), since we assume that W is inductive in S_0, and by our faithfulness restriction on expansion that

$$q_a \overset{s}{\underset{1}{\to}} q_b \Rightarrow \eta_1(q_a) \overset{s}{\underset{0}{\to}} \eta_1(q_b)$$

it follows then that $W(q_a)$ implies $W(q_b)$. In case 2), the truth of W is preserved by definition of W-preserving residuals. Case 3) follows from the assumption that the expansion is accurate and the argument of case 1).

THEOREM 3: If expansion $S_0 \xrightarrow[t]{} S_1$ is W-interleavable, then the expansion is W-consistent.

PROOF: Let $q_1 \varepsilon Q_1$ be such that $W(q_1)$ is true. Suppose that $q_1 \overset{y}{\underset{1}{\to}} q_2$, where $y \varepsilon \Sigma_1^*$. Since the expansion is extendable, y can be extended into a valid transition sequence x = yz such that

$$q_1 \overset{y}{\underset{1}{\to}} q_2 \overset{z}{\underset{1}{\to}} q_3$$

and q_3 is conceivable in S_i. By the definition of W-interleavability, x is W-compressible and hence there exists an x' performable in S_0 such that $q_1 \overset{x'}{\underset{1}{\to}} q_3$. By lemma 1 we have that $W(q_1)$ implies $W(q_3)$. Since we assume that W is true for the initial state, the theorem follows.

Let us consider some system, say S_0, in which the predicate W is inductive. If we expand S_0 by a W-interleavable expansion, then we have a W-consistent expansion, resulting in S_1. Now suppose that we continue to expand, creating systems $S_2, S_3, \ldots$, by W-consistent expansions. In order to show that the expansion $S_0 \xrightarrow[t]{} S_1 \xrightarrow[t]{} \ldots \xrightarrow[t]{} S_i$ is W-consistent, we would like to make use of the knowledge that the transitions of S_i are either transitions of S_0, or expansion transitions. Given a state q_1 of S_i, we wish there to be a valid transition sequence x such that $q_1 \overset{x}{\underset{1}{\to}} q_2$ and q_2 is conceivable in S_0. Using the ideas of W-commutativity, we then show that since W is inductive in S_0, $W(q_2)$ must be true in S_i.

In the definition of W-compressible, we were only concerned about single expansions. Now we are concerned about multi-expansions, and need to make certain that W-preserving residuals are not interleaved with expansion transitions of any system of the sequence, as they are only useful when applied to states that are conceivable in S_0.

We first broaden our definition of performable.

DEFINITION: Let $S_0 \xrightarrow[t]{} S_1 \xrightarrow[t]{} \ldots \xrightarrow[t]{} S_i$. A transition sequence x of S_i is performable in S_0 if

x is performable in S_{i-1},
$\mu_i(x)$ is performable in S_{i-2},
$\mu_{i-1}(\mu_i(x))$ is performable in S_{i-3},
⋮
$\mu_2(\mu_3(\ldots(\mu_i(x))\ldots))$ is performable in S_0.

We broaden our definition of W-compressibility to account for multi-expansions.

DEFINITION: A valid transition sequence x of S_i, where $S_0 \xrightarrow[t]{} S_1 \xrightarrow[t]{} \ldots \xrightarrow[t]{} S_i$, such that $q_1 \overset{x}{\underset{1}{\to}} q_2$ where q_1 and q_2 are conceivable in S_0, is said to be (0,W)-compressible if there exists a transition sequence $y = \ell_1 u \ell_2$ such that

1) u is from Σ_i^*,
2) ℓ_1 and ℓ_2 are W-preserving residuals,
3) $q_1 \overset{y}{\underset{i}{\rightarrow}} q_2$,
4) y is performable in S_0.

DEFINITION: An expansion is said to be (0,W)-interleavable if it is accurate, and each valid transition sequence starting from a state q such that q is conceivable in S_0 and W(q) is true can be S_0-extended so that the result is (0,W)-compressible.

In order to make use of the idea of W-commutativity in our new context, we need to choose the W-preserving residuals from a slightly restricted class. We will only want to apply W-preserving residuals to states that are conceivable in S_0, because only in these states will we know that W is true. So we will use those W-preserving residuals whose locations in a transition sequence can always be changed so that they only occur when the system is in a "proper" state.

DEFINITION: Let $S_0 \longrightarrow\!\!\!\!\rightarrow S_1 \longrightarrow\!\!\!\!\rightarrow \ldots \longrightarrow\!\!\!\!\rightarrow S_i$. A W-preserving residual ℓ is said to be 0-preadjusting in S_i if for all states q_1 and q_2 of Q_i, where q_1 is conceivable in S_0 and $W(q_1)$ is true, if

$$q_1 \overset{x\ell}{\underset{i}{\rightarrow}} q_2$$

where x is from Σ_i^*, then there exists a W-preserving residual ℓ_1 and a y from Σ_i^* such that

$$q_1 \overset{\ell_1 y}{\underset{i}{\rightarrow}} q_2$$

A W-preserving residual ℓ is said to be 0-postadjusting in S_i if for all states q_1, q_2, and q_3 of Q_i, where q_1 and q_3 are conceivable in S_0 and $W(q_1)$ is true

$$q_1 \overset{*}{\underset{i}{\rightarrow}} q_2 \overset{\ell x}{\underset{n}{\rightarrow}} q_3$$

where x is from Σ_i^*, then there exists a W-preserving residual ℓ_1 and a y from Σ_i^* such that

$$q_2 \overset{y\ell_1}{\underset{i}{\rightarrow}} q_3$$

Using these restricted W-preserving residuals, we extend our definition of W-commute.

DEFINITION: Let s be a transition of a process different from the expansion process, and let t_r be an expansion transition occurring in all expansion sequences. s is said to (0,W)-commute around t_r in S_i if

1) W is inductive in S_0, and
2) for each sequence $\langle t_1,\ldots,t_n\rangle$ from the set of expansion sequences T, all states q_1 and q_2 from Q_i such that q_1 is W-reachable, and for each $t_k \varepsilon \langle t_1,\ldots,t_n\rangle$,
   a) if t_k precedes t_r, then
   $$q_1 \overset{t_1 \ldots t_n s}{\underset{i}{\longrightarrow}} q_2 \Rightarrow q_1 \overset{\ell s t'_1 \ldots t'_j}{\underset{i}{\longrightarrow}} q_2$$
   where $\langle t'_1,\ldots,t'_j\rangle$ is a prefix of an expansion sequence, and ℓ is a 0-preadjusting W-preserving residual in S_{i-1},
   b) if t_r precedes t_k, then
   $$q_1 \overset{s t_k \ldots t_n}{\longrightarrow} q_2 \Rightarrow q_1 \overset{t'_k \ldots t'_m s \ell}{\underset{i}{\longrightarrow}} q_2$$
   where $\langle t'_k,\ldots,t'_m\rangle$ is a suffix of an expansion sequence, and ℓ is an 0-postadjusting W-preserving residual in S_{i-1}.

(0,W)-commutativity provides us with a sufficient condition for (0,W)-interleavability.

THEOREM 4: Let $S_0 \longrightarrow\!\!\!\!\rightarrow S_1 \longrightarrow\!\!\!\!\rightarrow \ldots \longrightarrow\!\!\!\!\rightarrow S_i$. If $S_0 \longrightarrow\!\!\!\!\rightarrow S_1 \longrightarrow\!\!\!\!\rightarrow \ldots \longrightarrow\!\!\!\!\rightarrow S_{i-1}$ is a (0,W)-interleavable expansion, and all transitions of processes other than the expansion process of S_i (0,W)-commute around t_r, for some expansion transition t_r, and the expansion $(S_0 \longrightarrow\!\!\!\!\rightarrow S_1 \longrightarrow\!\!\!\!\rightarrow \ldots \longrightarrow\!\!\!\!\rightarrow S_i)$ is accurate and extendable to S_0, then the expansion is (0,W)-interleavable.

PROOF: A straightforward induction based on the definition of (0,W)-commute.

Finally, we show that (0,W)-interleavability is a sufficient condition for W-consistency.

THEOREM 5: If W is inductive in S_0, and $S_0 \longrightarrow\!\!\!\!\rightarrow S_1 \longrightarrow\!\!\!\!\rightarrow \ldots \longrightarrow\!\!\!\!\rightarrow S_i$ is an (0,W)-interleavable expansion, then the expansion is W-consistent.

PROOF: Consider a state q_1 of S_i such that $W(q_1)$ is true. Let y be any valid transition sequence such that $q_1 \overset{y}{\underset{i}{\rightarrow}} q_2$. Since the expansion is (0,W)-interleavable, y can be S_0-extended to a valid transition sequence x = yz, such that

$$q_1 \overset{y}{\underset{i}{\rightarrow}} q_2 \overset{z}{\underset{i}{\rightarrow}} q_3$$

where q_3 is conceivable in S_0, and $z \varepsilon \Sigma_i^*$. We now proceed by induction on i. The basis, i = 1, is theorem 3. For the induction step, by definition of (0,W)-interleavability, x is (0,W)-compressible and hence there are exists an x' performable in S_0 of the form $\ell_1 u \ell_2$ such that

$$q_1 \overset{\ell_1}{\underset{i}{\rightarrow}} q'_1 \quad \overset{u}{\underset{i}{\rightarrow}} \quad q'_3 \overset{\ell_2}{\underset{i}{\rightarrow}} q_3$$

Here ℓ_1 and ℓ_2 are W-preserving residuals, and u is from Σ_i^*. By the restriction that

$$q_1 \overset{r}{\underset{i}{\rightarrow}} q_2 \Rightarrow \eta_i(q_1) \overset{r}{\underset{i-1}{\rightarrow}} \eta_i(q_2)$$

(faithfulness) and the assumption that the expansion is accurate, we have that

$$q_a \overset{u}{\underset{i-1}{\rightarrow}} q_b$$

where $q_a = \eta_i(q'_1)$ and $q_b = \eta_i(q'_3)$. From the induction hypothesis, we know that $W(q_a)$ implies $W(q_b)$. The theorem then follows since we know that ℓ_1 and ℓ_2 do not falsify W.

We have presented a method of "attacking the problem" of proving the correctness of a system involving complicated interaction among the processes. Simply stated, our method is to show that the interaction of processes can be ignored, thereby reducing a system with much interaction between processes to a simpler system with little interaction between processes. In the next section we illustrate our ideas with an example.

6. EXAMPLE - PARALLEL GARBAGE COLLECTION

In this section we use the theory which we have derived to develop and prove correct a program for parallel garbage collection. The program is basically that of Dijkstra et al. [4], although the correctness proof is ours.

What we envision is a LISP-like environment, with one process, called the mutator, performing basic LISP operations, i.e. manipulating a set of cells representing a graph structure, and another process, called the collector, performing garbage collection. The two processes will execute in parallel, with a "minimal amount of interaction". In particular, there will be no synchronizing places (i.e., neither process will ever have to wait for the other).

The variables used by the program include a finite set of cells. Each cell consists of three parts - a color and two pointers which will reference other cells. The two cells so referenced are called the left son and the right son. A subset of these cells is designated to be roots; they are always "accessible" by the two processes. Another cell is specialized to be the head of the free-list. The free-list is a list of garbage cells. A cell is termed garbage if it is not on a path from a root. It is termed mutatable otherwise. The collector process determines which cells are garbage, and then "places" these garbage cells on the free-list. The mutator process may take cells off the free-list, and make them sons of mutatable cells. For more details, see Dijkstra et al. [4].

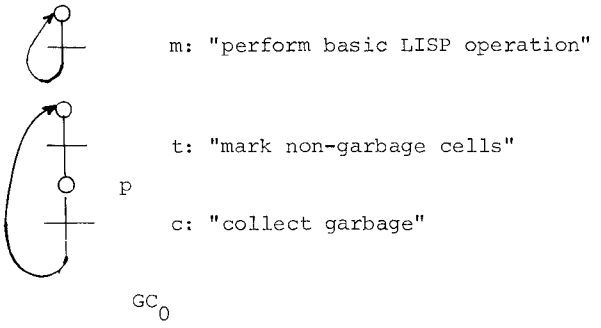


Figure 20.

Initially we consider the program of Figure 20, which we designate GC_0. The mutator process is conceptualized as one transition and one place. The transition will be thought of as performing some basic LISP operation.

The collector process is conceptualized as two transitions, representing a marking and a collecting phase, and two places. All cells are assumed to be initially white. The first transition colors all non-garbage cells black. The second transition then "places" all white (i.e. garbage) cells on the free-list, and resets the black cells to white.

We first sketch the expansions to GC_0; then we will prove them to be W-consistent, for a W yet to be presented.

The first step is to expand the "marking phase" of the collector. As is mentioned in Dijkstra et al. [4], it is necessary to specify some "overhead" on the part of the mutator. The reason for this is shown in Figure 21 (from [4]). In Figure 21a, root 2 points to cell a, and the collector is "examining" root 1 to "see" if it has any sons, which it does not. Next, as shown in Figure 21b, the collector examines root 2 to "see" if it has any sons. But in the meantime the mutator has modified the graph structure of the set of cells, and now root 1 points to cell a, but root 2 does not. The collector "sees" that root 2 does not have any sons, and incorrectly assumes that cell a is garbage.

In order to avoid this problem, we introduce an "intermediate color" for a cell - gray. The mutator, at the same instant that it establishes a pointer to a cell, makes the cell "at least gray", to use the term of Dijkstra et al. [4].

The details of the first expansion are shown in Figure 22. Here, "to shade a cell gray",

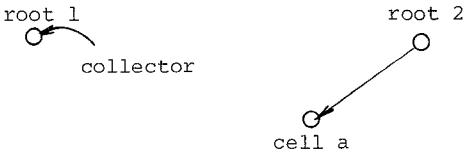


Figure 21a.

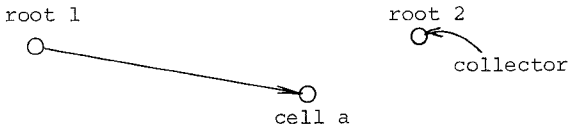
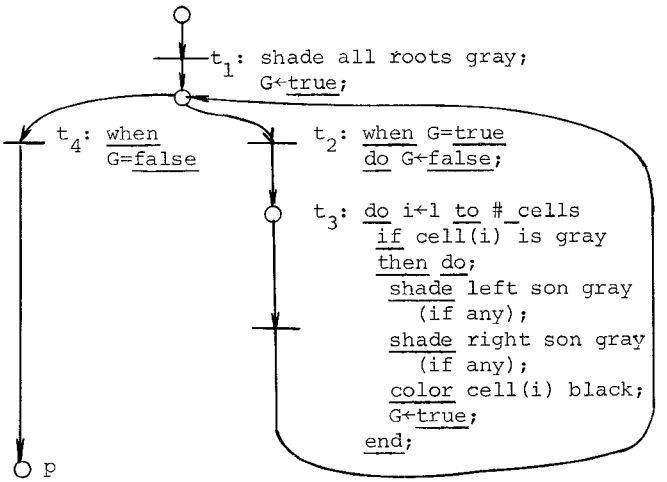
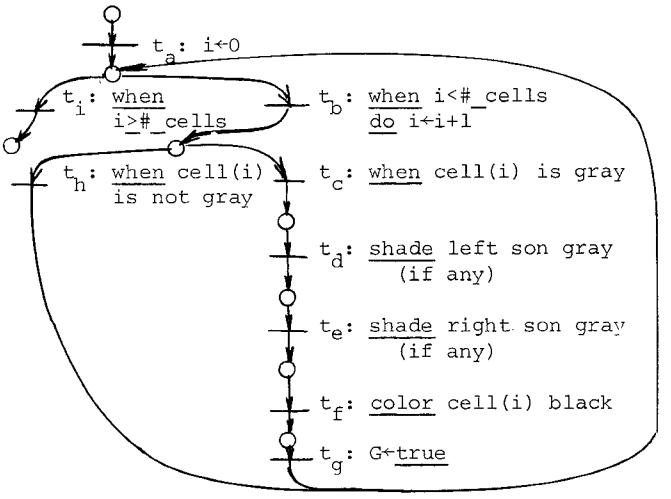


Figure 21b.



The expansion subgraph of $GC_0 \rightarrow\!\!\!\!\rightarrow GC_1$. This replaces transition t of GC_0 to form GC_1.

Figure 22.



The expansion subgraph of $GC_1 \rightarrow\!\!\!\!\rightarrow GC_2$.

Figure 23.

synonymous with "to make a cell at least gray", means

if the cell is white then color it gray;
else do nothing;

It is not difficult to verify that the expansion is accurate. We prove later that it is W-consistent as well.

The marking phase is further expanded in Figure

23. Now we come to the expansion of the mutator transition. We will ignore one detail - that of one process adding cells to the free-list while the other process removes them. This is an instance of the producer-consumer problem, which has been treated by several authors (see Dijkstra [2], for example).

There is one detail associated with the free-list that we do consider, however. This is the problem of how to handle a cell during the period from when it is taken off the free-list to when it is made the son of some mutatable cell. We solve this problem differently from as was done in Dijkstra et al. [4], where an elegant solution is presented to both this problem and the producer-consumer problem, but is not proven correct.

We have the mutator "mark" any cell that is about to be made the son of some other cell. This mark cannot be "erased" by the collector, and a marked cell, even if the cell is white, is not treated as garbage in the collection phase. The details of the expansion are shown in Figure 24.

The final expansion of the collecting phase, shown in Figure 25, is straightforward.

m_1: let c_1 be a mutatable cell
let c_2 be either a mutatable cell or a cell from the free-list
mark c_2

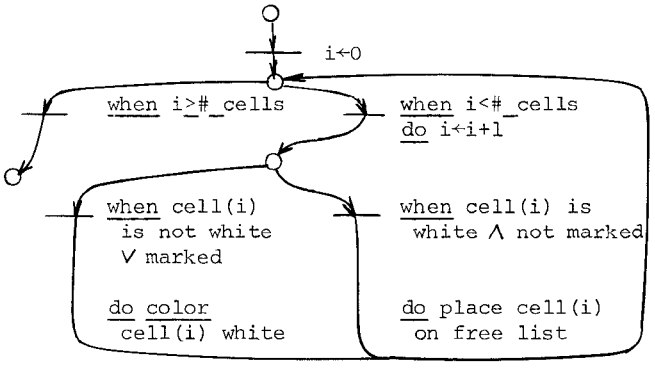
m_2: set c_1's right or left son to be c_2

m_3: shade c_2 gray

m_4: unmark c_2

The expansion subgraph of $GC_2 \longrightarrow\!\!\!\!\!\rightarrow GC_3$. Each cell has been given a "mark" field. To mark a cell means to set the field to 1, to unmark it means to set the field to 0. Transition c (the collection phase) is modified to read "place all cells that are white and unmarked on free-list".

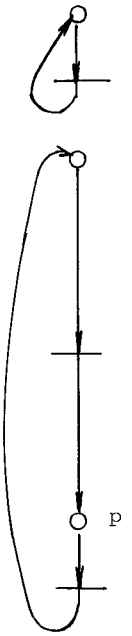
Figure 24.



The expansion subgraph of $GC_3 \longrightarrow\!\!\!\!\!\rightarrow GC_4$.

Figure 25.

We now concentrate on proving the parallel garbage collection program to be correct. GC_0 is rewritten in Figure 26 with more detail, but each



m: let c_1 be a mutatable cell.
let c_2 be either a mutatable cell or a cell from the free-list.
set c_1's right or left son to be c_2, and shade c_2 gray.

t: shade all roots gray
G←true;
do while (G);
G←false;
do i←1 to #_cells;
if cell(i) is gray then do;
shade left son gray (if any)
shade right son gray (if any)
color cell(i) black;
G←true;
end;
end;
end;

c: place all white cells on free-list;
color all cells white;

GC_0 with more detail.

Figure 26.

transition is still thought of as being indivisible. We write "black $\not\rightarrow$ white" to mean that no cell colored black has as either a right son or a left son a cell colored white, and similarly for "gray $\not\rightarrow$ white". We assume that in the initial state, all cells are colored white, and all cells except the roots are on the free list. The following predicates are easily shown to be inductive in GC_0.

1) black $\not\rightarrow$ white
2) i(collector) = p $\Rightarrow$ "all roots are colored black"
3) i(collector) = p $\Rightarrow$ gray $\not\rightarrow$ white

The conjunction of the three predicates, which we call W, implies that when the collector is at p, any white cell is garbage; however, they do not imply that all garbage cells are white.

We will modify GC_0 by a series of four expansions that are W-consistent. For the resulting system, GC_4, we will show that since the predicates are GC_0-inductive, there will always exist a reachable state in which the collector is at p and all and only all garbage cells are white.

The first expansion is shown in Figure 22. The key idea in showing that this expansion is W-interleavable is that if one considers a proper prefix of any expansion sequence, the composite effect of the expansion transitions, disregarding their effect on the collector's instruction pointer, is that of a W-preserving residual. This is easily verified by observing that the only predicate affected is "black $\not\rightarrow$ white". No cell is colored white in an expansion sequence, and a cell is colored black only if (simultaneously) its sons are shaded gray. (We note that in the above we are only considering the collector process, and hence can treat it as a sequential program, allowing us to examine the composite effect of expansion transitions.)

It follows that m, the mutator transition, W-commutes around the last occurrence of t_3 in any expansion sequence, i.e.

$$q_1 \xrightarrow[GC_1]{t_1(t_2t_3)^n m} q_2 \Rightarrow q_1 \xrightarrow[GC_1]{\ell_1 m t_1} q_2$$

and

$$q_1 \xrightarrow[GC_1]{t_1t_2(t_3t_2)^n m} q_2 \Rightarrow q_1 \xrightarrow[GC_1]{\ell_2 m t_1 t_2} q_2.$$

where n is a nonnegative integer, ℓ_1 is a W-preserving residual whose effect is that of $(t_2t_3)^n$ restricted to non-instruction pointer variables, and ℓ_2 is a W-preserving residual whose effect is that of $(t_3t_2)^n$ restricted to non-instruction pointer variables. The expansion is clearly accurate and extendable; hence by theorems 2 and 3 it is W-consistent.

The second expansion, resulting in GC_2, is shown in Figure 23. We want to show this expansion to be (0,W)-interleavable, as W is not inductive in GC_1.

We first show the expansion to be W-interleavable, i.e., if

$$q_1 \xrightarrow[GC_2]{t_a \ldots m \ldots m \ldots t_n} q_2 \text{ (the original sequence)}$$

then there exist W-preserving residuals ℓ_1 and ℓ_2 such that

$$q_1 \xrightarrow[GC_2]{\ell_1 t_a \ldots t_1 m m \ell_2} q_2 \text{ (the modified sequence)}$$

In order to show this, we need to determine the effect of moving the mutator transitions to the end of the expansion sequence. Any particular cell is either unaffected by the action of the occurrences of m or its components are changed in some way. The effect of an occurrence of m is either to change the value of one of a cell's pointers or to shade a cell gray.

We call the execution of either transition t_c or t_h visiting cell(i). If a cell is shaded gray by an occurrence of m before it is visited, then that cell will eventually be colored black and its sons shaded gray. In the modified sequence, the same effect can be achieved by having ℓ_1 shade the cell gray. If the cell is shaded gray by an occurrence of m after it is visited, then this shading will not cause other cells to be shaded by a transition of the expansion sequence. Hence in the modified sequence the effect can be achieved by having ℓ_2 shade the cell gray.

The modification of a pointer by m has two effects on a cell. The previous target of the pointer is "disowned", i.e. it becomes no longer a son of the changed cell. The new target of the pointer is "adopted", i.e. it becomes a son of the changed cell. The effect of delaying the adoption of a cell is the same as the effect of delaying the shading of a cell, which we have already discussed.

By delaying the disowning of a cell, we may be causing that cell (and its descendants) to be shaded when they were not shaded in the original sequence. We cannot account for this in ℓ_2, as this would require cells to be colored white, which would be difficult to show to be W-preserving. Instead we have ℓ_1 change any pointer that is modified by an occurrence of m to null. (This is permissible, as the actions of m are not predicated on the previous value of a pointer. It may be that in practice the previous value of a pointer is important for the execution of the mutator, but this value is not important in establishing the correctness of the garbage collection scheme.) The effects of this early disowning of a cell can then be handled by shading appropriate cells, as was previously discussed.

We now consider whether ℓ_1 and ℓ_2, as we have defined them, really are W-preserving residuals. That is, if W is true for some state, is W true in the state which is the image under ℓ_1 or ℓ_2? The only action of ℓ_2 is to shade some set of states gray. As the mutator's instruction pointer does not reference p, this will clearly preserve W. ℓ_1 will shade some set of cells gray, and will also set some set of pointers to null. Again, it is clear that neither of these actions will falsify W.

That the expansion is accurate and extendable is easily verifiable. Hence we have that it is W-interleavable. But we need to show that it is (0,W)-interleavable. This can be ascertained by the arguments which showed that the first expansion was W-interleavable, i.e., a proper prefix of an expansion sequence is effectively a W-preserving residual.

We now expand the mutator, resulting in GC_3, as shown in Figure 24. We will show that all transitions (0,W)-commute around m_2.

Actually, all transitions except c, the collection transition, commute around m_2. If c occurs immediately before m_3, the result is that the cell being shaded by the mutator at m_3 will end up shaded, but if c occurs after m_3, the result is that the cell will end up white. This can clearly be accounted for by a W-preserving residual ℓ after c that shades the appropriate cell gray.

It is trivial to verify that the other transitions commute around m_2. Hence we have that each collector transition W-commutes around m_2. The expansion is certainly accurate and extendable; hence it is W-interleavable. However, we still need to show it to be (0,W)-interleavable. We do this by showing that the W-preserving residual ℓ used in the expansion, i.e. shading a cell gray, is 0-postadjusting in GC_2. That is, we must show that if ℓ occurs in a state not conceivable in GC_0, then we can remove ℓ and replace it with a W-preserving residual ℓ', situated so that ℓ' will occur when the system is in a state conceivable in GC_0. Since the transition c and the prototype m of our current expansion exist in GC_0, ℓ will only occur in a state conceivable in GC_0. Hence the expansion is (0,W)-interleavable by theorem 4 and is W-consistent by theorem 5.

The final expansion, shown in Figure 25, is trivially W-consistent.

We have shown that our predicate W is GC_0-inductive in GC_4. As mentioned previously, this only implies that no non-garbage cells will be collected as garbage. It is possible that the system will never be in a state in which all garbage cells have been collected. But, whenever GC_4 reaches a state that is conceivable in GC_0, the predicate W will be true. From such a state, direct simulation of GC_0 is possible, and it is not difficult to prove that for any state of GC_0 in which W is true, after one execution of the marking transition, the

system will be in a state in which all and only all garbage cells are colored black. It then follows that GC_4 is semi-consistent.

Acknowledgements: We thank Professor Robert M. Keller for several helpful discussions about this paper. We also thank Mrs. Katrina Avery for typing the manuscript and Miss Eleanor Addison for drawing the figures.

7. REFERENCES

[1] Courtois, P.J., Heymans, F., and Parnas, D.L. Concurrent Control with "Readers" and "Writers". Communications of the ACM, vol. 14, no. 10 (October 1971).

[2] Dijkstra, E.W. "Notes on Structured Programming", in Structured Programming, by Dahl, Dijkstra, and Hoare, Academic Press (1972).

[3] Dijkstra, E.W., Lamport, L., Martin, A.J., Scholten, C.S., and Steffens, E.F.M. On-the-Fly Garbage Collection: an Exercise in Cooperation. Submitted to CACM, also EWD520 (October 1975).

[4] Doeppner, T.W., Jr., and Keller, R.M. On the Relevance of Abstract Models in Modeling Semaphore Implementations. Princeton University, Dept. of Electrical Engineering, Computer Science Laboratory Technical Report TR 193 (October 1975).

[5] Gries, D. On Structured Programming - A Reply to Smoliar. In ACM Forum, Communications of the ACM, vol. 17, no. 11 (November 1974)

[6] Infante, R., and Montanari, U. Proving Structured Programs Correct, Level by Level. Proceedings of 1975 International Conference on Reliable Software, SIGPLAN Notices, vol. 10, no. 6 (June 1975)

[7] Keller, R.M. Formal Verification of Parallel Programs. Communications of the ACM, vol. 19, no. 7 (July 1976)

[8] Lamport, L. On-the-Fly Garbage Collection: Once More with Rigor. Massachusetts Computer Associates CA-7508-1611 (August 1975).

[9] Lamport, L. Proving the Correctness of Multiprocessing Programs. Massachusetts Computer Associates CA-7508-0111 (August 1975).

[10] Lipton, R.J. Limitations of Synchronization Primitives with Conditional Branching and Global Variables. Proceedings of Sixth Annual ACM Symposium on Theory of Computing (April 1974).

[11] Lipton, R.J. Reduction: A Method of Proving Properties of Systems of Processes. Communications of the ACM, vol. 18, no. 12 (December 1975).

[12] Owicki, S., and Gries, D. Verifying Properties of Parallel Programs: an Axiomatic Approach. Communications of the ACM, vol. 19, no. 5 (May 1976).

[13] Rosen, B.K. Correctness of Parallel Programs: The Church-Rosser Approach. IBM Research Report RC5107 (October 1974).